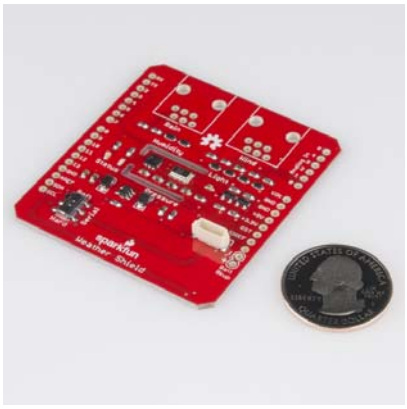
Weather Shield Hookup Guide

CONTRIBUTORS: *NATE*

♡ FAVORITE 3

Weather Shield Overview

The Weather Shield is an easy to use Arduino shield that grants you access to barometric pressure, relative humidity, luminosity, and temperature. There are also connections to optional sensors such as wind speed/direction, rain gauge, and GPS for location and super accurate timing.



The bare Weather Shield

Things you should know about this shield:

- Uses the HTU21D humidity sensor, MPL3115A2 barometric pressure sensor, and ALS-PT19 light sensor.
- Has connector for the GP-635T compact GPS module
- Has optional connectors for the SparkFun weather meters
- Weather shield can operate from 3V to 10V and has built in voltage regulators and signal translators
- Typical humidity accuracy of ±2%
- Typical pressure accuracy of ±50Pa
- Typical temperature accuracy of ±0.3C

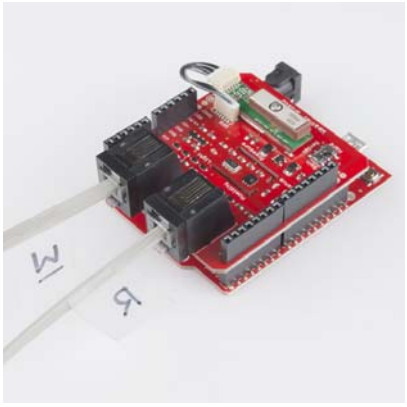
Suggested Reading

- I^2C Protocol
- Installing an Arduino library
- How to install Arduino shield headers
- What are pull-up resistors?
- HTU21D Humidity Sensor Hookup Guide
- MPL3115A2 Barometric Sensor Hookup Guide

Hooking It Up

To get up and running with the Weather Shield you'll need the following parts:

- Arduino, RedBoard, or other compatible board
- Arduino Stackable Headers
- Optional: GP-645T GPS Module and 1.75" mating cable
- Optional: Two RJ11 6-pin Connectors
- Optional: Weather Meters



Shield on a RedBoard with optional weather meter ('W'ind and 'R'ain cables) and GPS attached

Assembly

Solder the stackable headers onto the shield, and insert the shield into your Arduino. You are welcome to solder in the RJ11 connectors to the top of the board as well. If you have the GP-635T GPS module, don't worry about attaching it at this time, we'll get to GPS later.

Example Firmware - Basic

Using the Weather Shield example in the Arduino IDE relies on the HTU21D and MPL3115A2 libraries. As of Arduino v1.6.x you can download the libraries through the Arduino Library Manager. Search for and install “SparkFun MPL3115” and “SparkFun HTU21D”. For more information see our tutorial on using the Arduino library manager. For all the latest Arduino Weather Shield code, check out the Github Repository:

WEATHER SHIELD GITHUB REPO

Or copy and paste the code below into the Arduino IDE:

```
/*
 Weather Shield Example
 By: Nathan Seidle
 SparkFun Electronics
 Date: June 10th, 2016
 License: This code is public domain but you buy me a beer if you use this and we meet someday (Beerware license).

 This example prints the current humidity, air pressure, temperature and light levels.

 The weather shield is capable of a lot. Be sure to checkout the other more advanced examples for creating
 your own weather station.

 */

#include <Wire.h> //I2C needed for sensors
#include "SparkFunMPL3115A2.h" //Pressure sensor - Search "SparkFun MPL3115" and install from Library Manager
#include "SparkFunHTU21D.h" //Humidity sensor - Search "SparkFun HTU21D" and install from Library Manager

MPL3115A2 myPressure; //Create an instance of the pressure sensor
HTU21D myHumidity; //Create an instance of the humidity sensor

//Hardware pin definitions
//-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=
const byte STAT_BLUE = 7;
const byte STAT_GREEN = 8;

const byte REFERENCE_3V3 = A3;
const byte LIGHT = A1;
const byte BATT = A2;

//Global Variables
//-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=
long lastSecond; //The millis counter to see when a second rolls by

void setup()
{
  Serial.begin(9600);
  Serial.println("Weather Shield Example");

  pinMode(STAT_BLUE, OUTPUT); //Status LED Blue
  pinMode(STAT_GREEN, OUTPUT); //Status LED Green

  pinMode(REFERENCE_3V3, INPUT);
  pinMode(LIGHT, INPUT);

  //Configure the pressure sensor
  myPressure.begin(); // Get sensor online
  myPressure.setModeBarometer(); // Measure pressure in Pascals from 20 to 110 kPa
  myPressure.setOversampleRate(7); // Set Oversample to the recommended 128
  myPressure.enableEventFlags(); // Enable all three pressure and temp event flags

  //Configure the humidity sensor
  myHumidity.begin();

  lastSecond = millis();

  Serial.println("Weather Shield online!");
}

void loop()
{
  //Print readings every second
  if (millis() - lastSecond >= 1000)
  {
    digitalWrite(STAT_BLUE, HIGH); //Blink stat LED

    lastSecond += 1000;

    //Check Humidity Sensor
    float humidity = myHumidity.readHumidity();

    if (humidity == 998) //Humidty sensor failed to respond
    {
      Serial.println("I2C communication to sensors is not working. Check solder connections.");

      //Try re-initializing the I2C comm and the sensors
      myPressure.begin();
      myPressure.setModeBarometer();
      myPressure.setOversampleRate(7);
      myPressure.enableEventFlags();
      myHumidity.begin();
    }
    else
    {
      Serial.print("Humidity = ");
      Serial.print(humidity);
```

```
      Serial.print("%,");
      float temp_h = myHumidity.readTemperature();
      Serial.print(" temp_h = ");
      Serial.print(temp_h, 2);
      Serial.print("C,");

      //Check Pressure Sensor
      float pressure = myPressure.readPressure();
      Serial.print(" Pressure = ");
      Serial.print(pressure);
      Serial.print("Pa,");

      //Check tempf from pressure sensor
      float tempf = myPressure.readTempF();
      Serial.print(" temp_p = ");
      Serial.print(tempf, 2);
      Serial.print("F,");

      //Check light sensor
      float light_lvl = get_light_level();
      Serial.print(" light_lvl = ");
      Serial.print(light_lvl);
      Serial.print("V,");

      //Check batt level
      float batt_lvl = get_battery_level();
      Serial.print(" VinPin = ");
      Serial.print(batt_lvl);
      Serial.print("V");

      Serial.println();
    }

    digitalWrite(STAT_BLUE, LOW); //Turn off stat LED
  }

  delay(100);
}

//Returns the voltage of the light sensor based on the 3.3V rail
//This allows us to ignore what VCC might be (an Arduino plugged into USB has VCC of 4.5 to 5.2V)
float get_light_level()
{
  float operatingVoltage = analogRead(REFERENCE_3V3);

  float lightSensor = analogRead(LIGHT);

  operatingVoltage = 3.3 / operatingVoltage; //The reference voltage is 3.3V

  lightSensor = operatingVoltage * lightSensor;

  return (lightSensor);
}

//Returns the voltage of the raw pin based on the 3.3V rail
//This allows us to ignore what VCC might be (an Arduino plugged into USB has VCC of 4.5 to 5.2V)
//Battery level is connected to the RAW pin on Arduino and is fed through two 5% resistors:
//3.9K on the high side (R1), and 1K on the low side (R2)
float get_battery_level()
{
  float operatingVoltage = analogRead(REFERENCE_3V3);

  float rawVoltage = analogRead(BATT);

  operatingVoltage = 3.30 / operatingVoltage; //The reference voltage is 3.3V

  rawVoltage = operatingVoltage * rawVoltage; //Convert the 0 to 1023 int to actual voltage on BATT pin

  rawVoltage *= 4.90; //(3.9k+1k)/1k - multiple BATT voltage by the voltage divider to get actual system voltage

  return (rawVoltage);
}
```

Open the Serial Monitor. You should see the following output:

```
Humidity = 28.12%, temp_h = 29.54C, Pressure = 84051.25Pa, temp_p = 82.85F, light_lvl = 0.96V, VinPin = 4.40V
Humidity = 28.36%, temp_h = 29.54C, Pressure = 84052.50Pa, temp_p = 82.96F, light_lvl = 0.96V, VinPin = 4.40V
Humidity = 28.46%, temp_h = 29.56C, Pressure = 84050.25Pa, temp_p = 82.96F, light_lvl = 0.96V, VinPin = 4.40V
Humidity = 28.24%, temp_h = 29.54C, Pressure = 84051.25Pa, temp_p = 82.96F, light_lvl = 0.96V, VinPin = 4.40V
```

Put your hand over the small clear device labeled ‘Light’ and watch the light level change to 0. Blow lightly on the humidity sensor and watch the humidity change.

Troubleshooting

If there is an error you will see:

```
I2C communication to sensors is not working. Check solder connections.
```

This message appears when the board is unable to get a response from the I2C sensors. This could be because of a faulty solder connection, or if there are other devices on the A5/A4 lines (which are also called SDA/SCL).

Example Firmware - Weather Station

For the more adventurous, we have the **Weather Station** example. This code demonstrates *all* the bells and whistles of the shield. You will need a weather station hooked up to see the wind speed, wind direction and rain values change.

```
/*
 Weather Shield Example
 By: Nathan Seidle
 SparkFun Electronics
 Date: November 16th, 2013
 License: This code is public domain but you buy me a beer if you use this and we meet someday (Beerware license).

 Much of this is based on Mike Grusin's USB Weather Board code: https://www.sparkfun.com/products/10586

 This is a more advanced example of how to utilize every aspect of the weather shield. See the basic
 example if you're just getting started.

 This code reads all the various sensors (wind speed, direction, rain gauge, humidty, pressure, light, batt_lvl)
 and reports it over the serial comm port. This can be easily routed to an datalogger (such as OpenLog) or
 a wireless transmitter (such as Electric Imp).

 Measurements are reported once a second but windspeed and rain gauge are tied to interrupts that are
 calcualted at each report.

 This example code assumes the GPS module is not used.

 */

#include <Wire.h> //I2C needed for sensors
#include "SparkFunMPL3115A2.h" //Pressure sensor - Search "SparkFun MPL3115" and install from Library Manager
#include "SparkFunHTU21D.h" //Humidity sensor - Search "SparkFun HTU21D" and install from Library Manager

MPL3115A2 myPressure; //Create an instance of the pressure sensor
HTU21D myHumidity; //Create an instance of the humidity sensor

//Hardware pin definitions
//-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=
// digital I/O pins
const byte WSPEED = 3;
const byte RAIN = 2;
const byte STAT1 = 7;
const byte STAT2 = 8;

// analog I/O pins
const byte REFERENCE_3V3 = A3;
const byte LIGHT = A1;
const byte BATT = A2;
const byte WDIR = A0;
//-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=

//Global Variables
//-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=
long lastSecond; //The millis counter to see when a second rolls by
byte seconds; //When it hits 60, increase the current minute
byte seconds_2m; //Keeps track of the "wind speed/dir avg" over last 2 minutes array of data
byte minutes; //Keeps track of where we are in various arrays of data
byte minutes_10m; //Keeps track of where we are in wind gust/dir over last 10 minutes array of data

long lastWindCheck = 0;
volatile long lastWindIRQ = 0;
volatile byte windClicks = 0;

//We need to keep track of the following variables:
//Wind speed/dir each update (no storage)
//Wind gust/dir over the day (no storage)
//Wind speed/dir, avg over 2 minutes (store 1 per second)
//Wind gust/dir over last 10 minutes (store 1 per minute)
//Rain over the past hour (store 1 per minute)
//Total rain over date (store one per day)

byte windspdavg[120]; //120 bytes to keep track of 2 minute average

#define WIND_DIR_AVG_SIZE 120
int winddiravg[WIND_DIR_AVG_SIZE]; //120 ints to keep track of 2 minute average
float windgust_10m[10]; //10 floats to keep track of 10 minute max
int windgustdirection_10m[10]; //10 ints to keep track of 10 minute max
volatile float rainHour[60]; //60 floating numbers to keep track of 60 minutes of rain

//These are all the weather values that wunderground expects:
int winddir = 0; // [0-360 instantaneous wind direction]
float windspeedmph = 0; // [mph instantaneous wind speed]
float windgustmph = 0; // [mph current wind gust, using software specific time period]
int windgustdir = 0; // [0-360 using software specific time period]
float windspdmph_avg2m = 0; // [mph 2 minute average wind speed mph]
int winddir_avg2m = 0; // [0-360 2 minute average wind direction]
float windgustmph_10m = 0; // [mph past 10 minutes wind gust mph ]
int windgustdir_10m = 0; // [0-360 past 10 minutes wind gust direction]
float humidity = 0; // [%]
float tempf = 0; // [temperature F]
float rainin = 0; // [rain inches over the past hour)] -- the accumulated rainfall in the past 60 min
volatile float dailyrainin = 0; // [rain inches so far today in local time]
```

```
//float baromin = 30.03;// [barom in] - It's hard to calculate baromin locally, do this in the agent
float pressure = 0;
//float dewptf; // [dewpoint F] - It's hard to calculate dewpoint locally, do this in the agent

float batt_lvl = 11.8; //[analog value from 0 to 1023]
float light_lvl = 455; //[analog value from 0 to 1023]

// volatiles are subject to modification by IRQs
volatile unsigned long raintime, rainlast, raininterval, rain;

//-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=

//Interrupt routines (these are called by the hardware interrupts, not by the main code)
//-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=
void rainIRQ()
// Count rain gauge bucket tips as they occur
// Activated by the magnet and reed switch in the rain gauge, attached to input D2
{
    raintime = millis(); // grab current time
    raininterval = raintime - rainlast; // calculate interval between this and last event

    if (raininterval > 10) // ignore switch-bounce glitches less than 10mS after initial edge
    {
        dailyrainin += 0.011; //Each dump is 0.011" of water
        rainHour[minutes] += 0.011; //Increase this minute's amount of rain

        rainlast = raintime; // set up for next event
    }
}

void wspeedIRQ()
// Activated by the magnet in the anemometer (2 ticks per rotation), attached to input D3
{
    if (millis() - lastWindIRQ > 10) // Ignore switch-bounce glitches less than 10ms (142MPH max reading) after the reed switch closes
    {
        lastWindIRQ = millis(); //Grab the current time
        windClicks++; //There is 1.492MPH for each click per second.
    }
}


void setup()
{
    Serial.begin(9600);
    Serial.println("Weather Shield Example");

    pinMode(STAT1, OUTPUT); //Status LED Blue
    pinMode(STAT2, OUTPUT); //Status LED Green

    pinMode(WSPEED, INPUT_PULLUP); // input from wind meters windspeed sensor
    pinMode(RAIN, INPUT_PULLUP); // input from wind meters rain gauge sensor

    pinMode(REFERENCE_3V3, INPUT);
    pinMode(LIGHT, INPUT);

    //Configure the pressure sensor
    myPressure.begin(); // Get sensor online
    myPressure.setModeBarometer(); // Measure pressure in Pascals from 20 to 110 kPa
    myPressure.setOversampleRate(7); // Set Oversample to the recommended 128
    myPressure.enableEventFlags(); // Enable all three pressure and temp event flags

    //Configure the humidity sensor
    myHumidity.begin();

    seconds = 0;
    lastSecond = millis();

    // attach external interrupt pins to IRQ functions
    attachInterrupt(0, rainIRQ, FALLING);
    attachInterrupt(1, wspeedIRQ, FALLING);

    // turn on interrupts
    interrupts();

    Serial.println("Weather Shield online!");

}

void loop()
{
    //Keep track of which minute it is
  if(millis() - lastSecond >= 1000)
    {
        digitalWrite(STAT1, HIGH); //Blink stat LED

    lastSecond += 1000;
```

```
        //Take a speed and direction reading every second for 2 minute average
        if(++seconds_2m > 119) seconds_2m = 0;

        //Calc the wind speed and direction every second for 120 second to get 2 minute average
        float currentSpeed = get_wind_speed();
        windspeedmph = currentSpeed; //update global variable for windspeed when using the printWeather() function
        //float currentSpeed = random(5); //For testing
        int currentDirection = get_wind_direction();
        windspdavg[seconds_2m] = (int)currentSpeed;
        winddiravg[seconds_2m] = currentDirection;
        //if(seconds_2m % 10 == 0) displayArrays(); //For testing

        //Check to see if this is a gust for the minute
        if(currentSpeed > windgust_10m[minutes_10m])
        {
            windgust_10m[minutes_10m] = currentSpeed;
            windgustdirection_10m[minutes_10m] = currentDirection;
        }

        //Check to see if this is a gust for the day
        if(currentSpeed > windgustmph)
        {
            windgustmph = currentSpeed;
            windgustdir = currentDirection;
        }

        if(++seconds > 59)
        {
            seconds = 0;

            if(++minutes > 59) minutes = 0;
            if(++minutes_10m > 9) minutes_10m = 0;

            rainHour[minutes] = 0; //Zero out this minute's rainfall amount
            windgust_10m[minutes_10m] = 0; //Zero out this minute's gust
        }

        //Report all readings every second
        printWeather();

        digitalWrite(STAT1, LOW); //Turn off stat LED
    }

  delay(100);
}

//Calculates each of the variables that wunderground is expecting
void calcWeather()
{
    //Calc winddir
    winddir = get_wind_direction();

    //Calc windspeed
    //windspeedmph = get_wind_speed(); //This is calculated in the main loop on line 179

    //Calc windgustmph
    //Calc windgustdir
    //These are calculated in the main loop

    //Calc windspdmph_avg2m
    float temp = 0;
    for(int i = 0 ; i < 120 ; i++)
        temp += windspdavg[i];
    temp /= 120.0;
    windspdmph_avg2m = temp;

    //Calc winddir_avg2m, Wind Direction
    //You can't just take the average. Google "mean of circular quantities" for more info
    //We will use the Mitsuta method because it doesn't require trig functions
    //And because it sounds cool.
    //Based on: http://abelian.org/vlf/bearings.html
    //Based on: http://stackoverflow.com/questions/1813483/averaging-angles-again
    long sum = winddiravg[0];
    int D = winddiravg[0];
    for(int i = 1 ; i < WIND_DIR_AVG_SIZE ; i++)
    {
        int delta = winddiravg[i] - D;

        if(delta < -180)
            D += delta + 360;
        else if(delta > 180)
            D += delta - 360;
        else
            D += delta;

        sum += D;
    }
```

```
    winddir_avg2m = sum / WIND_DIR_AVG_SIZE;
    if(winddir_avg2m >= 360) winddir_avg2m -= 360;
    if(winddir_avg2m < 0) winddir_avg2m += 360;

    //Calc windgustmph_10m
    //Calc windgustdir_10m
    //Find the largest windgust in the last 10 minutes
    windgustmph_10m = 0;
    windgustdir_10m = 0;
    //Step through the 10 minutes
    for(int i = 0; i < 10 ; i++)
    {
        if(windgust_10m[i] > windgustmph_10m)
        {
            windgustmph_10m = windgust_10m[i];
            windgustdir_10m = windgustdirection_10m[i];
        }
    }

    //Calc humidity
    humidity = myHumidity.readHumidity();
    //float temp_h = myHumidity.readTemperature();
    //Serial.print(" TempH:");
    //Serial.print(temp_h, 2);

    //Calc tempf from pressure sensor
    tempf = myPressure.readTempF();
    //Serial.print(" TempP:");
    //Serial.print(tempf, 2);

    //Total rainfall for the day is calculated within the interrupt
    //Calculate amount of rainfall for the last 60 minutes
    rainin = 0;
    for(int i = 0 ; i < 60 ; i++)
        rainin += rainHour[i];

    //Calc pressure
    pressure = myPressure.readPressure();

    //Calc dewptf

    //Calc light level
    light_lvl = get_light_level();

    //Calc battery level
    batt_lvl = get_battery_level();
}

//Returns the voltage of the light sensor based on the 3.3V rail
//This allows us to ignore what VCC might be (an Arduino plugged into USB has VCC of 4.5 to 5.2V)
float get_light_level()
{
    float operatingVoltage = analogRead(REFERENCE_3V3);

    float lightSensor = analogRead(LIGHT);

    operatingVoltage = 3.3 / operatingVoltage; //The reference voltage is 3.3V

    lightSensor = operatingVoltage * lightSensor;

    return(lightSensor);
}

//Returns the voltage of the raw pin based on the 3.3V rail
//This allows us to ignore what VCC might be (an Arduino plugged into USB has VCC of 4.5 to 5.2V)
//Battery level is connected to the RAW pin on Arduino and is fed through two 5% resistors:
//3.9K on the high side (R1), and 1K on the low side (R2)
float get_battery_level()
{
    float operatingVoltage = analogRead(REFERENCE_3V3);

    float rawVoltage = analogRead(BATT);

    operatingVoltage = 3.30 / operatingVoltage; //The reference voltage is 3.3V

    rawVoltage = operatingVoltage * rawVoltage; //Convert the 0 to 1023 int to actual voltage on BATT pin

    rawVoltage *= 4.90; //(3.9k+1k)/1k - multiple BATT voltage by the voltage divider to get actual system voltage

    return(rawVoltage);
}

//Returns the instataneous wind speed
float get_wind_speed()
{
    float deltaTime = millis() - lastWindCheck; //750ms
```

```
    deltaTime /= 1000.0; //Covert to seconds

    float windSpeed = (float)windClicks / deltaTime; //3 / 0.750s = 4

    windClicks = 0; //Reset and start watching for new wind
    lastWindCheck = millis();

    windSpeed *= 1.492; //4 * 1.492 = 5.968MPH

    /* Serial.println();
     Serial.print("Windspeed:");
     Serial.println(windSpeed);*/

    return(windSpeed);
}

//Read the wind direction sensor, return heading in degrees
int get_wind_direction()
{
    unsigned int adc;

    adc = analogRead(WDIR); // get the current reading from the sensor

    // The following table is ADC readings for the wind direction sensor output, sorted from low to high.
    // Each threshold is the midpoint between adjacent headings. The output is degrees for that ADC reading.
    // Note that these are not in compass degree order! See Weather Meters datasheet for more information.

    if (adc < 380) return (113);
    if (adc < 393) return (68);
    if (adc < 414) return (90);
    if (adc < 456) return (158);
    if (adc < 508) return (135);
    if (adc < 551) return (203);
    if (adc < 615) return (180);
    if (adc < 680) return (23);
    if (adc < 746) return (45);
    if (adc < 801) return (248);
    if (adc < 833) return (225);
    if (adc < 878) return (338);
    if (adc < 913) return (0);
    if (adc < 940) return (293);
    if (adc < 967) return (315);
    if (adc < 990) return (270);
    return (-1); // error, disconnected?
}


//Prints the various variables directly to the port
//I don't like the way this function is written but Arduino doesn't support floats under sprintf
void printWeather()
{
    calcWeather(); //Go calc all the various sensors

    Serial.println();
    Serial.print("$,winddir=");
    Serial.print(winddir);
    Serial.print(",windspeedmph=");
    Serial.print(windspeedmph, 1);
    Serial.print(",windgustmph=");
    Serial.print(windgustmph, 1);
    Serial.print(",windgustdir=");
    Serial.print(windgustdir);
    Serial.print(",windspdmph_avg2m=");
    Serial.print(windspdmph_avg2m, 1);
    Serial.print(",winddir_avg2m=");
    Serial.print(winddir_avg2m);
    Serial.print(",windgustmph_10m=");
    Serial.print(windgustmph_10m, 1);
    Serial.print(",windgustdir_10m=");
    Serial.print(windgustdir_10m);
    Serial.print(",humidity=");
    Serial.print(humidity, 1);
    Serial.print(",tempf=");
    Serial.print(tempf, 1);
    Serial.print(",rainin=");
    Serial.print(rainin, 2);
    Serial.print(",dailyrainin=");
    Serial.print(dailyrainin, 2);
    Serial.print(",pressure=");
    Serial.print(pressure, 2);
    Serial.print(",batt_lvl=");
    Serial.print(batt_lvl, 2);
    Serial.print(",light_lvl=");
    Serial.print(light_lvl, 2);
    Serial.print(",");
    Serial.println("#");
```

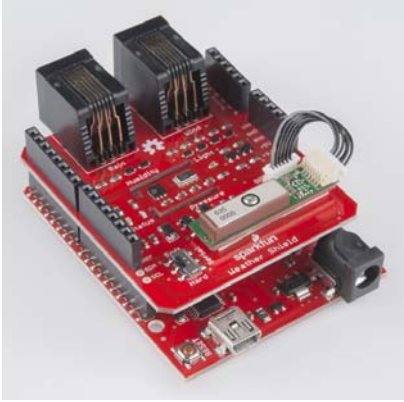
```
}
```

Open the Serial Monitor, and you should see an output string every second containing the current weather information:

```
$,winddir=0,windspeedmph=0,windspdmph_avg2m=0.0,winddir_avg2m=0,windgustmph_10m=0.0,windgustdir_10m=0,humidity=31.7,tempf=76.3,rainin=0.00,dail
yrainin=0.00,pressure=81525.25,batt_lvl=4.32,light_lvl=2.03,#

$,winddir=0,windspeedmph=0,windspdmph_avg2m=0.0,winddir_avg2m=0,windgustmph_10m=0.0,windgustdir_10m=0,humidity=31.7,tempf=76.3,rainin=0.00,dail
yrainin=0.00,pressure=81520.75,batt_lvl=4.32,light_lvl=2.02,#

$,winddir=0,windspeedmph=0,windspdmph_avg2m=0.0,winddir_avg2m=0,windgustmph_10m=0.0,windgustdir_10m=0,humidity=31.7,tempf=76.3,rainin=0.00,dail
yrainin=0.00,pressure=81517.50,batt_lvl=4.34,light_lvl=2.11,#

$,winddir=0,windspeedmph=0,windspdmph_avg2m=0.0,winddir_avg2m=0,windgustmph_10m=0.0,windgustdir_10m=0,humidity=31.7,tempf=76.3,rainin=0.00,dail
yrainin=0.00,pressure=81509.25,batt_lvl=4.31,light_lvl=2.11,#
```

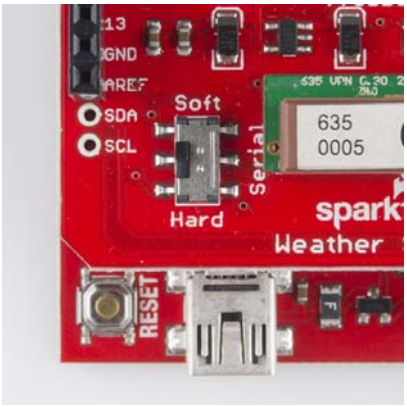
The `$` and `#` are start and stop characters. These types of bytes are used to make it easy to parse out the data. For example, you could have an Electric Imp listen for a `$` and record the data until you see a `#`. Once you have the string then split on the commas (also known as comma delimited), and start recording the next string.

Example with GPS



Shield on a RedBoard with optional weather meter connectors and GPS attached

Attach the GP-635T GPS module using the short cable. To secure the module, there is space on the shield to attach the module using double-stick tape.



Serial pins are connected to digital pins 4 and 5 when Serial is set to soft and are attached to the internal UART when set to hard.

There is a switch labeled **Serial** on the shield. This is to select which pins on the Arduino to connect the GPS to. In almost all cases the switch should be set to 'Soft'. This will attach the GPS serial pins to digital pins 5 (TX from the GPS) and 4 (RX into the GPS).

Grab the GPS example sketch that demonstrates using the GP-635T with all the other sensors. Load it onto your Arduino, and open the serial monitor at 9600. You should see output similar to the following:

```
$,winddir=-1,windspeedmph=nan,humidity=28.2,tempf=76.1,rainin=0.00,dailyrainin=0.00,pressure=81355.00,batt_lvl=4.05,light_lvl=3.05,lat=40.01805
4,lat=-105.282577,altitude=1647.40,sats=10,date=11/16/2013,time=20:00:44,#

$,winddir=-1,windspeedmph=nan,humidity=28.2,tempf=76.1,rainin=0.00,dailyrainin=0.00,pressure=81358.00,batt_lvl=4.07,light_lvl=3.05,lat=40.01805
4,lat=-105.282577,altitude=1647.40,sats=10,date=11/16/2013,time=20:00:45,#

$,winddir=-1,windspeedmph=nan,humidity=28.2,tempf=76.1,rainin=0.00,dailyrainin=0.00,pressure=81358.25,batt_lvl=4.08,light_lvl=3.05,lat=40.01805
4,lat=-105.282585,altitude=1647.40,sats=10,date=11/16/2013,time=20:00:46,#
```

Note: The batt_lvl is indicating 4.08V. This is correct and is the actual voltage read from the Arduino powered over USB. The GPS module will add 50-80mA to the overall power consumption. If you are using a long or thin USB cable you may see significant voltage drop similar to this example. There is absolutely no harm in this! The Weather Shield runs at 3.3V and the Arduino will continue to run just fine down to about 3V. The reading is very helpful for monitoring your power source (USB, battery, solar, etc).

This example demonstrates how you can get location, altitude, and time from the GPS module. This would be helpful with weather stations that are moving such as balloon satellites, AVL, package tracking, and even static stations where you need to know precise altitude or timestamps.

Resources and Going Further

The Weather Shield example firmware outputs regular barometric pressure. This is very different from the pressure that weather stations report. For more information, see the definition of “altimeter setting pressure”. For an example of how to calculate altimeter setting type barometric pressure see the MPL3115A2 hook-up guide. Also checkout the MPL3115A2 library, specifically the `BarometricHgInch` example.

Datasheets

There’s a lot of technology on this shield. Here’s the datasheets in case you need to reference them:

- HTU21D Humidity
- MPL3115A2 Pressure
- ALS-PT19 Light
- GP-635T GPS
- Weather Meters

Additional resources and projects to checkout:

- HTU21D Humidity Repo and Library
- MPL3115A2 Pressure Repo and Library
- If you’re interested in using GPS with Arduino definitely checkout Mikal Hart’s TinyGPS++ library
- Consider adding an OpenLog for datalogging the weather readings over time
- Electric Imp is a good way to add WiFi to get a truly wireless weather station